## Deployment detail

The backend DRF and the PostgresSQL database are running in two containers respectively with their dependencies configured by the docker-compose.yml. The react front end website is built and deployed statically onto the nginx server on AWS EC2. Here is the IP of the website http://18.116.70.118/. (I didn't purchase a domain)

## Features

### Token-based list sharing

With token-based list sharing, the shared list gains an access token. By sharing the sharing URL appended with the access token of the list, your friends or family members can see the grocery list you make without registering an account. Meanwhile, as the account owner, you can stop the sharing of a list whenever you want. The reason I decided to do token-based list sharing is because an URL can be accessed easily by any devices that can connect to the internet (The device needs to have a display as well so that you can see the list). I didn't set expiration of a list but let users decide when they want to stop sharing as a grocery list is not a sensitive document that can only be viewed within a period of time.

### JWT authentication

By implementing JWT authentication for the backend, it is easier to scale the backend authentication services with more servers and more clusters because it is stateless. Compared to a stateful session approach, we don't have to worry about the query database repetitively to store session data. Hence, it reduces database IO and makes response time faster. Meanwhile, it has a number of other benefits like it fits well with microservices architecture, works better on a Single-Page-Application, Cross-Domain Authentication, etc., which we can talk about in detail in our later interviews.

## Color Selection for the list

Yes! I try to make the list creating process interesting by creating a row of colors for the user to choose from. If you regret it, you can always change to the one you want later by editing the list.

## Possible Improvements

A lot of these improvements were not made when I was writing the project was due to insufficient time. However, with more time given, I will make the following improvements.

### Front End Improvement (React)

1. Since there were only a few days for writing, testing, and deploying the app, I didn't have time to design the component relationship perfectly before I could start writing the code. Hence, there are lots of components that can be parametrized and reused, especially those buttons I have in my project. I reused the <AuthForm … /> component for both login and register such that I don't have to repeat JSX code, but as I have mentioned, there could be more carefully designed reusable components to reduce code redundancy and increase maintainability.
2. The second thing to improve is to use the refresh token to retrieve a new access token (jwt token) for users to make their browsing experience more continuous. In my case, I only used the access token, and once it expires, the user will be prompted to login again.
3. I was initially planning to improve authentication experience by incorporating third-party sign-in, but unfortunately, I did not have enough time. However, if I did, this is definitely a must-have for a modern application.

4. The color scheme and page structure can be designed better using some prototyping tool like Figma.
5. Consider using Next.js for better SEO support or seeking other libraries or frameworks that can improve the SEO of the website.
6. Supposedly speaking the shared link of the list should be sent to the email or the phone of the user such that it is easier for the user to share it with others (Copy and paste into Facebook, WhatsApp, Instagram etc.) However, due to the time limit, I toggled an alert box for the user to see the shared link. Don't worry, you can always check them by clicking the shared list button and remove those you no longer want to share.
7. I should have done some data preprocessing before submitting the registration or login data to the backend because the email of user credentials is case-sensitive, and I think this is not right. Always need to normalize the data before submitting it to the backend.
8. A more reasonable component structures and relations need to be designed.
9. Front end form needs validation. (Is the format of the mail valid?)
10. Write more comments.

## Backend Improvement (General, not limited to DRF)

1. I should implement password reset functionality and email verifications when registering.
2. The generation of the token used for list sharing should be tossed into celery workers such that if there are millions of users creating tokens, we can still handle it easily. Since celery uses RabbitMQ, I will not mention the use of MQ to improve response time later.
3. Authentication module, token generation and distribute module (data sharing module), and database servers can be grouped into different clusters, using RPC or other communication protocols to exchange data and services. Meanwhile, we need to provide service discovery and registration cluster for registering these microservices we broke down (ZooKeeper, Eureka, Consul)
4. We also need to have an API gateway for rate-limiting, routing, security, and other features to make our website more robust and efficient.
5. Logging. This includes using ELK stack for analyzing and logging the system, which can help us find problems faster if there are any.
6. Meanwhile, if the IO of database is pressured hard, consider adding a cache cluster using Redis before accessing data in the database (ex. hotspot on twitter, tweets of celebrities are more likely to be viewed by people). I think the challenge here is what to cache? (What do we use for key, and what values does the key correspond to? Does using a long query string as the key hinder the performance of the querying of searching the cache?
7. If the amount of data in a database grows, we should consider database sharding, this brings benefits but also concerns if we have distributed transactions in our system. However, I think the benefits beat the concerns when our application is data intensive.